Cairo University
Faculty of Engineering
Department of Computer Engineering

Inferex

A Graduation Project Report Submitted

to

Faculty of Engineering, Cairo University

in Partial Fulfillment of the requirements of the degree

of

Bachelor of Science in Computer Engineering.

Presented by
Aly Abdelmotaleb

Supervised by
Dr. Sandra Wahid

July 2025

Abstract

Inferex is an intelligent desktop application designed to help users recall and retrieve information they have previously seen on their computers whether from a video, document, or website without the need to manually save or organize it. In today's digital world, users interact with large volumes of information every day, yet remembering specific details from past activities remains a constant challenge. Most AI assistants currently available require users to manually provide the context of the question in order to receive an accurate and personalized answer, which limits their convenience and effectiveness. Inferex solves this problem by silently and privately building a local context from the user's screen activity, enabling it not only to retrieve previously seen content but also to answer related questions intelligently without requiring additional input from the user. The system operates entirely offline, ensuring complete privacy by keeping all data on the device. It captures screenshots periodically, segments them into meaningful parts, and stores them in a vector database for semantic retrieval. Lightweight algorithms and compressed models are used to guarantee smooth performance even on personal devices with limited hardware resources. Inferex offers a private, context-aware, and searchable memory of the user's digital interactions, making it a powerful tool for productivity, learning, and everyday information recall.

الملخص

إنفريكس (Inferex) هو تطبيق ذكي لسطح المكتب صُمم لمساعدة المستخدمين على تذكّر واسترجاع المعلومات التي شاهدوها سابقًا على أجهزتهم، سواء كانت من فيديو، مستند، أو موقع إلكتروني، دون الحاجة إلى حفظها أو تنظيمها يدويًا. في عالمنا الرقمي اليوم، يتفاعل المستخدمون يوميًا مع كميات ضخمة من المعلومات، ومع ذلك فإن تذكّر التفاصيل الدقيقة من الأنشطة السابقة لا يزال تحديًا مستمرًا. تعتمد معظم أدوات الذكاء الاصطناعي المتاحة حاليًا على أن يقوم المستخدم بتوفير سياق السؤال يدويًا للحصول على إجابة دقيقة وشخصية، مما يقلل من كفاءتها ويجعل استخدامها أقل سهولة. يحلّ إنفريكس هذه المشكلة من خلال إنشاء سياق محلي تلقائي وخاص بناءً على نشاط الشاشة لدى المستخدم، مما يمكّنه ليس فقط من استرجاع المحتوى الذي تم عرضه سابقًا، بل أيضًا من الإجابة على الأسئلة المتعلقة به بذكاء ودون الحاجة إلى إدخال أي معلومات إضافية. يعمل النظام بشكل كامل دون اتصال بالإنترنت، مما يضمن خصوصية المستخدم من خلال إبقاء جميع البيانات محفوظة على الجهاز فقط. يلتقط التطبيق لقطات شاشة بشكل دوري، ويقوم بتقسيمها إلى أجزاء ذات معنى، ثم يخزنها في قاعدة بيانات يمكن البحث فيها بطريقة دلالية. كما تم استخدام خوارزميات خفيفة ونماذج مضغوطة لضمان أداء سلس حتى على الأجهزة الشخصية ذات الموارد المحدودة. يقدم إنفريكس ذاكرة رقمية خاصة، مدركة للسياق، وقابلة للبحث، مما يجعله أداة قوية لزيادة الإنتاجية، وتسهيل التعلّم، واسترجاع المعلومات في الحياة اليومية.

ACKNOWLEDGMENT

I would like to thank my twin brother Mahmoud Abdelmotaleb for his support through our journey in Cairo university faculty of engineering. May God have mercy on him.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

Abbreviation	Full Form
CLAHE	Contrast Limited Adaptive Histogram Equalization
CV	Computer Vision
IoU	Intersection over Union
LLM	Large Language Model
mAP	Mean Average Precision
NMS	Non-Maximum Suppression
R-CNN	Region-based Convolutional Neural Network
RPN	Region Proposal Network
RoI	Region of Interest

Contacts

Team Members

Name	Email	Phone Number
Aly Abdelmotaleb	aly.mf.2001@gmail.com	+201024205330

Supervisor

Name	Email	Number
Sandra Wahid	sandrawahid@hotmail.com	-

This page is left intentionally empty

Chapter 1: Introduction

In Inferex, my contributions spanned four main modules: Screen segmentation part using 2 approaches (Squeezedet with the implementation of its loss function along with FasterRcnn using resnet50 as backbone), 2 classical models for the text alignment part, Activity Finder, data preprocessing & backend for the project

1.1. Document Organization

This section outlines the structure of the report and provides a brief overview of the upcoming chapters. Chapter 2 presents a detailed explanation of my contributions to the modules I worked on. Chapter 3 discusses the methods used to test and evaluate these contributions. Chapter 4 offers a brief conclusion, summarizing the experience I gained and the challenges I encountered throughout the project.

Chapter 2:System Design and Architecture

In this chapter, I provide a detailed explanation of my contributions to the project.

2.1. Screen Segmentation Module: SqueezeDet

2.1.1. Functional Description

The SqueezeDet-based Screen Segmentation module performs real-time detection of image and text regions within a screenshot using a lightweight, single-stage detector. The module inputs a preprocessed screenshot and outputs bounding-box coordinates and class labels for each detected region (image vs. text).

2.1.2. Modular Decomposition

1. **Model Loading**
    - Load the pruned SqueezeDet checkpoint (≈9 MB) into memory.
    - Initialize the network architecture with pretrained convolutional weights and custom detection heads.
2. **Preprocessing**
    - Resize the input screenshot to match the network's input resolution (e.g., 768×1024).
    - Normalize pixel values using the mean/std from ImageNet.
3. **Forward Pass**
    - Pass the image tensor through the convolutional backbone (SqueezeNet layers).
    - Generate feature maps shared by objectness, bounding-box, and classification heads.

4. **Anchor Matching & Inference**
    - Compute predicted offsets and objectness scores per anchor box.
    - Apply non-maximum suppression (NMS) to filter overlapping detections (IoU threshold = 0.5).
    - Map predictions back to original image coordinates.
5. **Postprocessing**
    - Separate detections into two lists: `text` and `image` based on the class score.
    - Format output as a list of dictionaries: `{"class": ..., "bbox": [x1,y1,x2,y2], "score": ...}`.

2.1.3. Loss Function

With no public SqueezeDet implementation available, we re-implemented the training loss from scratch:

1. **Objectness Loss (L_{obj}):** Binary cross-entropy between predicted objectness scores and ground-truth anchor assignments.
2. **Bounding-Box Regression Loss (L_{reg}):** Smooth L_1 loss on predicted offsets (t_x, t_y, t_w, t_h) for positive anchors.
3. **Class Loss (L_{cls}):** Cross-entropy over detected classes per positive anchor.

$$L = \lambda_{\text{obj}} L_{\text{obj}} + \lambda_{\text{reg}} L_{\text{reg}} + \lambda_{\text{cls}} L_{\text{cls}}$$

Figure 2.1: Overall training Loss function

2.2 Screen Segmentation Module: Faster R-CNN (ResNet-50)

2.2.1. Functional Description

This two-stage detector uses a Region Proposal Network (RPN) atop a ResNet-50 backbone to generate high-quality region proposals, then refines and classifies each proposal into `text` or `image` categories with precise localization.

2.2.2. Modular Decomposition

1. **Backbone Feature Extraction**
    - Feed the screenshot through ResNet-50 up to conv4 stage.
    - Produce high-resolution feature maps.
2. **Region Proposal Network**
    - Slide small convolution over feature maps to predict objectness and anchor offsets.
    - Generate top-N region proposals by selecting anchors with highest objectness scores and applying NMS.
3. **RoI Pooling & Head Processing**
    - Crop and resize each proposal to a fixed spatial size via RoIAlign.
    - Pass through the detection head (two fully connected layers) to output:
        - Multi-class probabilities (background, text, image).
        - Bounding-box deltas for each class.

4. **Inference & NMS**
    - Discard proposals classified as background.
    - Apply class-wise NMS on remaining boxes (IoU > 0.5).
    - Format final detections as `{class, bbox, score}`.

2.2.3. Loss Function

The multi-task loss sums RPN and Fast R-CNN components:

$$L_{\text{RPN}}(\{p_i\}, \{t_i\}) = \frac{1}{N_{\text{cls}}} \sum_i L_{\text{cls}}(p_i, p_i^*) + \frac{\lambda}{N_{\text{reg}}} \sum_i p_i^* \, L_{\text{reg}}(t_i, t_i^*)$$

Figure 2.2 : RPN Loss function

$$L_{\text{RCNN}}(p, u, t_u, v) = -\log p_u + 1_{u \geq 1} \, L_{\text{reg}}(t_u, v)$$

Figure 2.3 :Classification and bounding box regression loss

2.3. Text Detection Module: Classical CV-Based Detector (Legacy)

2.3.1. Functional Description

An initial, pure computer-vision pipeline that locates text regions through binarization, morphology, and contour filtering. Retained for benchmarking and comparative analysis only.

2.3.2. Modular Decomposition

1. **Grayscale & Binarization**
    - Convert to grayscale.
    - Apply Kassar binarization, then Otsu thresholding.

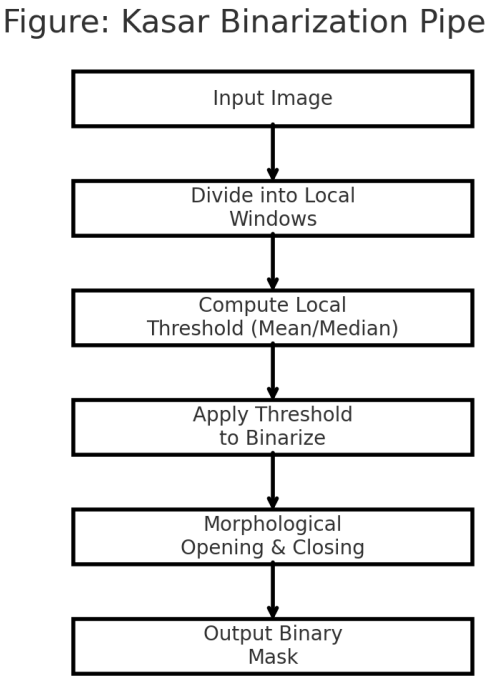

Figure 2.4: Kassar binarization

2. **Morphological Filtering**
    - Perform opening/closing to remove small artifacts and bridge text strokes.
3. **Edge Detection & Contour Extraction**
    - Run Canny edge detector.
    - Find external contours.
4. **Geometric Filtering**
    - Discard contours with aspect ratio or area outside thresholds.
    - Filter by width ≥ 6 px, height 5–240 px.
5. **Bounding-Box Assembly**
    - Compute bounding boxes for remaining contours.
    - Merge overlapping boxes based on IoU (>0.3).

2.3.3. Limitations

- **F1 Score:** < 0.50.
- **Adaptability:** Poor across font sizes/styles.

2.4. Text Detection Module: Heuristic-Enhanced Detector

2.4.1. Functional Description

A robust, multi-stage heuristic pipeline combining advanced preprocessing, ensemble thresholding, and geometry-based filtering to extract high-quality text regions from masked or noisy screenshots.

2.4.2. Modular Decomposition

1. **Advanced Preprocessing**
    - CLAHE (grid=18×18, clip=1.5).
    - Bilateral filtering for edge preservation.
    - Dynamic parameter scaling based on local contrast.
2. **Multi-Variant Binarization**
    - Ensemble of: Adaptive Mean, Adaptive Gaussian, Global Otsu, and inverted variants.
3. **Text-Specific Morphology**
    - Vertical dilation (1×3 kernel) to link broken characters.
    - Character bridging to reconnect fragmented strokes.
4. **Geometric Filtering**
    - Retain boxes: width ≥ 6 px, height 5–240 px, aspect ratio 0.27–15.5, fill ratio 25–86 %.

5. **Grouping & Conflict Resolution**
    - Horizontal merging for gaps ≤ 15 px.
    - Vertical separation to maintain line integrity.
    - Resolve overlaps via IoU-based containment logic.
6. **Real-Time Tuning**
    - OpenCV trackbars for 15+ parameters.

2.4.3. Performance Summary

- **F1 Score:** ≈ 0.55.
- **Compute:** Slightly higher than classical CV but acceptable on CPU.

2.5. Activity Finder Module

2.5.1. Functional Description

The Activity Finder identifies the user's current context—browser, local video, or other applications—to annotate screenshots with relevant metadata (URLs, file paths, app names).

2.5.2. Modular Decomposition

1. **Active Window Identification**
    - Use Windows API (`ctypes`) to retrieve foreground window handle, process ID, and title.

2. **Process Classification**
    - **Browser Activity**
        - Detect known browser processes (Chrome, Firefox, Edge).
        - Use `pywinauto` to extract the current URL from the address bar.
    - **Video Activity**
        - Enumerate processes via `psutil` to find VLC, MPC-HC, PotPlayer.
        - Parse command-line arguments for video file paths.
    - **Generic Application**
        - Fallback to application executable name and window title when no browser/video is detected.
3. **Utility Submodules**
    - URL normalization & validation.
    - File-path cleaning & normalization.

2.5.3. Design Constraints

- **Accuracy:** Must correctly identify URL or file path ≥ 95 % of the time.
- **Performance:** Execution under 100 ms per check to avoid UI lag.
- **Compatibility:** Support multiple browser versions and media players via conditional logic and fallbacks.

2.6 Backend Service Module

2.6.1. Functional Description

The Backend Service module is implemented as a FastAPI application providing a RESTful API that orchestrates database operations, model inference, and background tasks. It initializes the FastAPI app, mounts routes with dependency injection for database sessions and model interfaces, and leverages asynchronous endpoints for streaming LLM responses. The API layer handles chat management, extractive QA, LLM queries, text processing, and image handling, while coordinating daemon and processing threads for screenshots and data pipelines.

2.6.2. Modular Decomposition

API Layer (FastAPI)

- **Chat Endpoints:** Create and retrieve chats and messages (`/chats/`, `/chats/{id}`)
- **QA Endpoint:** `/ask/eqa` performs extractive QA via a fine-tuned model
- **LLM Endpoint:** `/ask/llm` streams LLM-generated responses
- **Text Processing:** `/test/textProcessing` and `/test/textProcessing/search` for custom text ingestion and retrieval
- **Image Handling:** `/image`, `/image/{filename}` to list, fetch, and delete screenshots
- **Settings Management:** `/settings/` (GET and PUT) for daemon and model configuration
- **Erase Endpoint:** `/erase` clears database, deletes files, and resets system state

Database Models & Schemas

- **SQLAlchemy Models:** Represent `Chat`, `Message`, and `Setting`, with relationships and JSON-based context fields
- **Pydantic Schemas:** Define `Chat`, `ChatWithMessages`, `Message`, `ContextItem`, `Settings`, and various request/response formats

Background Daemons

- **Screenshot Daemon:** Periodically captures screenshots using `mss`, applies SSIM-based change detection, and stores new images
- **Image Processing Thread:** Handles segmentation and annotation of new screenshots
- **Data Processing Thread:** Transforms extracted content via a text processing interface and stores embeddings in ChromaDB

Model Inference Interfaces

- **Extractive QA:** Uses a fine-tuned BERT pipeline
- **LLM Models:** Runs compressed Qwen3 model via DFloat11 and Quantized Qwen3 model via LLaMA-based inference
- **Embedding:** Implements `Text2Embedding` and `ImageEmbedder` modules for vector-based search in ChromaDB

Utilities & Middleware

- **CORS Middleware:** Enables cross-origin requests
- **Thumbnail & Base64 Conversion:** Prepares screenshot previews
- **Settings Initialization:** Automatically sets default configurations on system startup

Chapter 3: System Testing and Verification

In this chapter, I will demonstrate how I tested and verified the correctness and effectiveness of my contribution.

3.1. Testing Setup

The testing was conducted on a laptop with the following specifications:

- CPU: AMD Ryzen 7 5800H
- RAM: 16 GB
- GPU: NVIDIA RTX 3060
- OS: Windows 11
- Python version: 3.10

3.2. Testing Plan and Strategy

For the AI-based modules, the strategy involves splitting the data into training and validation sets, monitoring the loss on both, and selecting the model version that achieves low validation loss without signs of overfitting.

In the **Activity Finder** module, I used extensive debugging with custom print statements to trace the behavior of the system at each step, which was crucial for handling varied application interfaces. I tested numerous use cases—including different browsers, video players, and unsupported apps—to ensure accurate detection of user activity. This helped refine the logic for extracting URLs and file paths, validate edge cases, and verify the correctness of outputs in real time.

3.2.1. Module Testing

3.2.1.1. Screen Segmentation (SqueezeDet and Faster R-CNN):

For evaluating segmentation performance, I used **Mean Average Precision (mAP)** with an **IoU threshold of 0.5**. SqueezeDet was tested using the anchors generated via k-means clustering, but it exhibited low precision for the image class. Faster R-CNN, with a ResNet-50 backbone, yielded more reliable results and was selected as the primary model.

In both cases, we used a dataset of **25,000 annotated screenshots**, ensuring a consistent evaluation protocol across models.

Training loss	result
Squeezedet	**Text only** → 4.135
FasterRcnn resnet 50	0.3028

Table 3.1: Training loss for SqueezeDet & FasterRcnn

Validation mAP	result
Squeezedet	Best epoch is 100 Both classes → 38.3% **Text only** → 53.8%
FasterRcnn resnet 50	Best epoch is 7 Image → 64.6% , Text → 74.3% mAP → 69.5%

Table 3.2: mAP Validation for SqueezeDet & FasterRcnn

test mAP	result
FasterRcnn resnet 50	Image → 55.2% , Text → 73.2% mAP → 64.2%

Table 3.3: mAP Testing for FasterRcnn

3.2.1.2. Text Identification:

The effectiveness of the text region extraction was also assessed using mAP at an IoU threshold of 0.5. Multiple image processing techniques were iteratively tested to optimize alignment, and the final pipeline achieved a maximum mAP of 0.55.

3.2.1.3. Activity Finder:

Functionality was validated by manually enabling debug outputs to trace execution at each stage. I tested a wide range of real-world use cases across different browsers, video players, and unsupported applications to confirm the accurate extraction of URLs or file paths and ensure consistent behavior across platforms.

3.3. Testing Schedule

Each component was tested immediately upon completion to verify its functionality and correctness before moving on to the next stage.

Chapter 4: Conclusions and Faced Challenges

4.1. Faced Challenges

During the development of my work in the screen segmentation module, the main challenges included data scarcity for training and dataset annotation. The changes made to the SqueezeDet module introduced complexities in the screen segmentation part. After extensive searching and iterations, I managed to find a dataset that partially addressed our problem, although it lacked annotations. To address this, I performed preprocessing and generated anchor boxes using k-means clustering, selecting the optimal number of anchors by plotting the elbow curve—this setup was later used for training SqueezeDet. Training SqueezeDet and Faster R-CNN posed further challenges, requiring multiple trials due to low precision, particularly in image class segmentation using SqueezeDet. As a result, I primarily utilized Faster R-CNN for segmentation. In the text identification section, the initial attempt with the pseudo code from the 'ScreenSeg' paper failed, possibly due to differences in text size between smartphones and desktops. Another approach involved extensive image processing, yielding an improved Mean Average Precision (MAP) of 0.55, yet it was insufficient to significantly enhance the quality of Faster R-CNN results. Additionally, in the activity finder, besides retrieving activities, I also had to extract file paths or URLs and verify their validity, testing various libraries until achieving success in this aspect.

4.2. Gained Experience

Through my work on the project at Inferex, I gained hands-on experience across multiple technical areas. In the **screen segmentation module**, I deepened my understanding of object detection models by working with both **SqueezeDet**—where I implemented and customized its loss function and optimized anchor box generation using **k-means clustering and elbow method**—and **Faster R-CNN** with a **ResNet-50 backbone**, handling various challenges related to training stability and precision.

In the **text identification module**, I explored classical computer vision approaches and learned the limitations of relying solely on published pseudo code, adapting my pipeline with advanced image processing techniques to achieve measurable performance improvements.

The **activity finder** module allowed me to develop robust **parsing and validation pipelines**, experimenting with multiple libraries to reliably extract file paths or URLs and confirm their validity.

4.3. Conclusions

This phase of the project provided valuable insights into the practical challenges of building a screen segmentation and activity analysis system. Despite data limitations and architectural complexities, I was able to iteratively refine the pipeline through data preprocessing, model experimentation, and tool integration. The transition from SqueezeDet to Faster R-CNN, the application of classical vision techniques for text detection, and the development of reliable activity parsing workflows all contributed to a deeper understanding of end-to-end system design. Overall, this experience strengthened my ability to apply both classical and deep learning approaches to solve real-world problems in a modular and scalable way.

References

[1] B. Wu, A. Wan, F. Iandola, P. H. Jin, and K. Keutzer, “SqueezeDet: Unified, Small, Low Power Fully Convolutional Neural Networks for Real-Time Object Detection for Autonomous Driving,” arXiv preprint arXiv:1612.01051, 2019.

[2] M. Goyal, D. Garg, R. S. Munjal, D. P. Mohanty, S. Moharana, and S. P. Thota, “ScreenSeg: On-Device Screenshot Layout Analysis,” arXiv preprint arXiv:2104.08052, 2021.

[3] T. Kasar, J. Kumar, and A. G. Ramakrishnan, “Font and Background Color Independent Text Binarization,” in *proceedings of the Ninth International Conference on Document Analysis and Recognition (ICDAR)*, Curitiba, Brazil, Sep. 23-26, 2007, pp. 838–842.